МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ОТЧЕТ ПО ИГРЕ**

**«YELLOW SUBMARINE»**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_М.М.Шадрина

Направление подготовки 02.03.03 — «Математическое обеспечение и

администрирование информационных систем»\_\_\_\_\_\_\_\_\_\_\_курс\_\_\_\_\_3\_\_\_\_\_

Направленность (профиль) Технология программирования

Научный руководитель

канд физ.-мат. наук, доцент,                                                            О.В. Гаркуша

Краснодар

2024

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_Toc185807557)

[2 Описание объектов сцены 4](#_Toc185807558)

[3 Описание функций на C# 5](#_Toc185807559)

[3.1 ObjectSpawner 5](#_Toc185807560)

[3.2 MouseClickHandler 5](#_Toc185807561)

[3.3 Movement 6](#_Toc185807562)

[4 Текст программы 7](#_Toc185807563)

# 1 Постановка задачи

Целью данного проекта является разработка интерактивной игры, вдохновленной серией мультфильмов «Смешарики», а именно эпизодом «Балласт». Основная идея игры заключается в создании игрового процесса, где игрок управляет клешней, установленной на лодке, чтобы собирать призы, избегая препятствий в виде плавающих ботинок.

Основные задачи проекта:

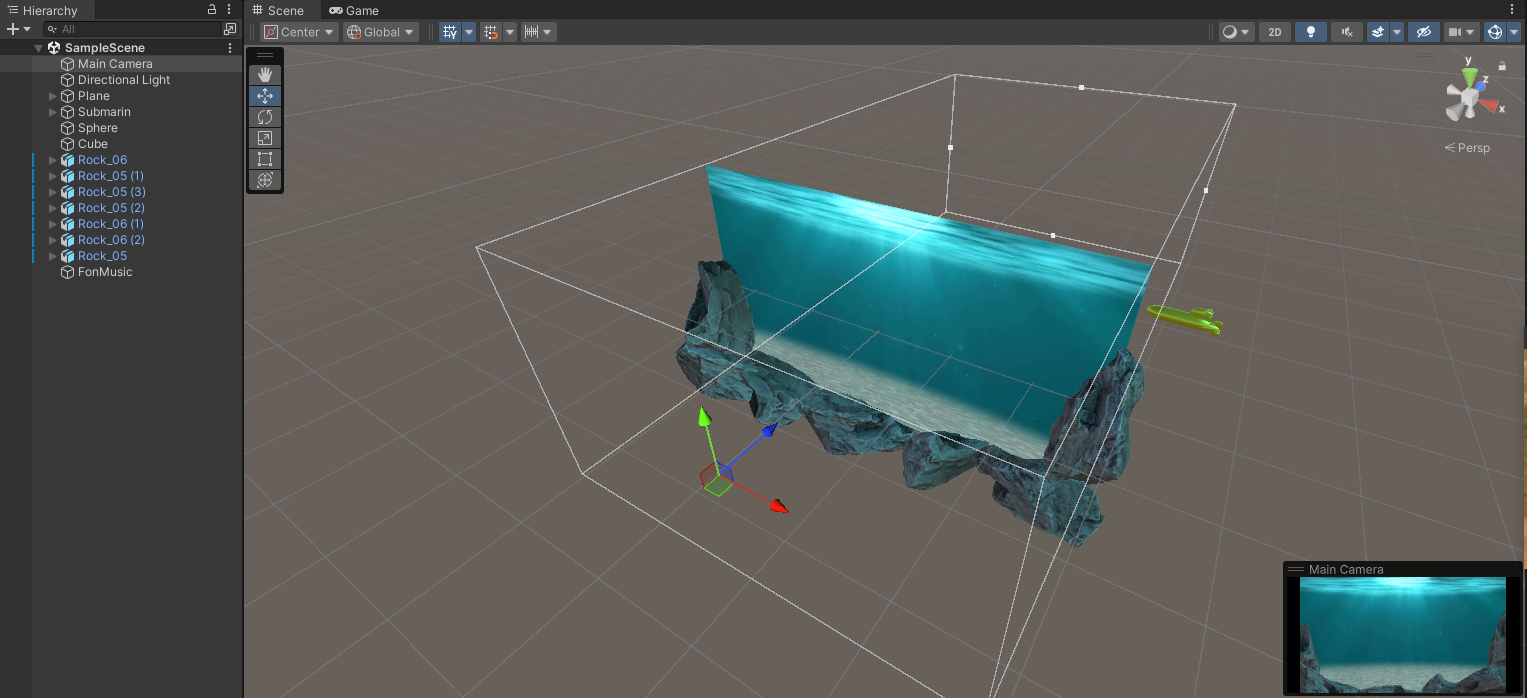
1. Разработать игровую механику управления клешней, позволяющую игроку взаимодействовать с объектами на сцене.
2. Реализовать физическое поведение объектов в воде, включая движение субмарины.
3. Реализовать движение винтов выплывающей субмарины.
4. Добавить фон и окружение сцены.
5. Реализовать движение субмарины.
6. Создать визуальное оформление анимации пузырьков в воде появляющихся у винтов.
7. Интегрировать звуковое сопровождение в виде песни Yellow Submarine - The Beatles.

Игра должна быть простой в освоении, динамичной и соответствовать задумке оригинального мультфильма. Разработка будет реализована с использованием Unity и языка программирования C#.

# 2 Описание объектов сцены

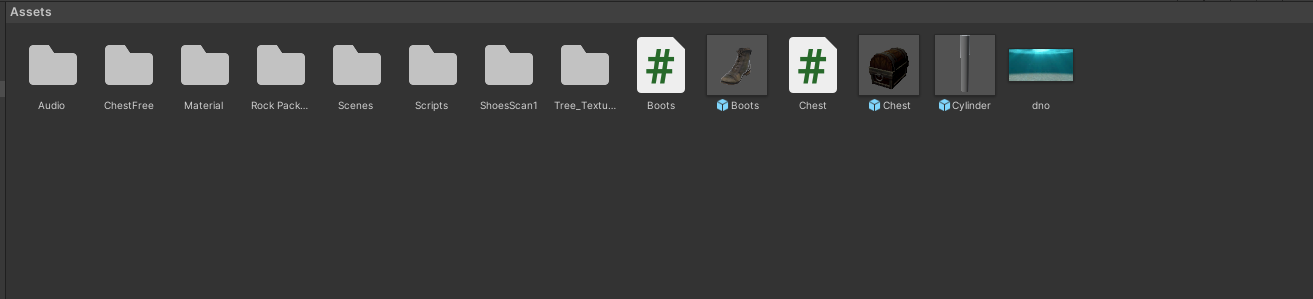
Сцена содержит следующие объекты:

* Main Camera - камера.
* Directional Light - объект освещения.
* Plane - задний фон.
* Submarin - группа объектов в форме субмарины.
* Sphere и Cube для получения координат при программировании.
* FonMusic - объект вечно играющий музыку Yellow Submarine - The Beatles.
* Несколько объектов Rock из готовой бесплатной библиотеки .



В качестве префабов хранятся слдующие элементы:

* Boots из готовой бесплатной библиотеки
* Chest из готовой бесплатной библиотеки
* Cylinder



# 3 Описание функций на C#

3.1 ObjectSpawner

1. **Start()**
   * Запускает корутину для создания объектов с заданной задержкой.
   * Устанавливает маску слоя камешков ROCKS.
2. **SpawnObjectsWithDelay()**
   * Ожидает указанное время задержки.
   * Проверяет наличие префаба и создает объекты ботинок (botinockPrefab) в пределах видимости камеры.
   * Учитывает минимальное расстояние между объектами и избегает пересечения с подводной лодкой или камешками, используя физические лучи (Physics.Raycast).
   * Добавляет случайный поворот для созданных объектов, если это разрешено.

3.2 MouseClickHandler

1. **Start()**
   * Устанавливает маски слоёв камешков и объектов по умолчанию.
   * Запускает корутину, разрешающую клики через указанное время.
2. **Update()**
   * Проверяет нажатие левой кнопки мыши.
   * При нажатии запускает луч из камеры в мировое пространство и обрабатывает попадания.
   * Если это первый клик, строит цилиндр от субмарины к точке попадания. Для последующих кликов строит цилиндры от последней точки.
3. **EnableClicksAfterDelay(float delay)**
   * Корутина, которая включает возможность обрабатывать клики после указанной задержки.
4. **CreateCylinder(Vector3 startPoint, Vector3 endPoint)**
   * Создает цилиндр между двумя точками.
   * Проверяет пересечение с камнями с помощью Physics.Raycast и завершает выполнение, если препятствие найдено.
   * Настраивает положение, направление, длину и ширину цилиндра.
   * Если на пути встречается объект с компонентом Chest, окрашивает цилиндры в зеленый цвет (успех). Если объект содержит компонент Boots, удаляет цилиндры, окрашивает их в красный цвет (неудача), и сбрасывает данные.
5. **Colorize(Color color)**
   * Изменяет цвет всех созданных материалов.

3.3 Movement

1. **Start()**
   * Получает компонент Rigidbody объекта.
2. **FixedUpdate()**
   * Рассчитывает направление от текущей позиции объекта к целевой позиции.
   * Перемещает объект к целевой позиции с учетом заданной скорости.
   * Ограничивает перемещение так, чтобы объект не проходил мимо цели.

# 4 Текст программы

Исходники по адресу:

<https://github.com/ShadrinaM/Unity_project_Yellow_Submarine>

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ObjectSpawner : MonoBehaviour

{

// Префаб для создания объектов Botinock

public GameObject botinockPrefab;

// Время задержки перед созданием объектов

public float delayTime = 4f;

// Общее количество объектов для создания

public int objectCount = 6;

// Минимальное расстояние между объектами

public float minDistance = 400f;

// Поворачивать ли?

public bool Povorot = true;

[Min(0)] public float mashtab = 1;

public Transform Submarin;

//Слой камешков

private LayerMask RockMask;

// Список для хранения позиций созданных объектов

private List<Vector3> spawnedPositions = new List<Vector3>();

private bool hasSpawned = false;

void Start()

{

RockMask = LayerMask.GetMask("ROCKS");

// Запускаем корутину для создания объектов с задержкой

StartCoroutine(SpawnObjectsWithDelay());

}

private IEnumerator SpawnObjectsWithDelay()

{

// Ждем указанное время

yield return new WaitForSeconds(delayTime);

// Если объекты уже были созданы, не создаем их снова

if (hasSpawned)

{

yield break;

}

// Проверяем, что префаб задан

if (botinockPrefab == null)

{

Debug.LogError("Botinock prefab is not assigned!");

yield break;

}

// Получаем камеру

Camera mainCamera = Camera.main;

// Получаем координаты углов видимости камеры

Vector3 bottomLeft = mainCamera.ViewportToWorldPoint(new Vector3(0, 0, 0));

Vector3 topRight = mainCamera.ViewportToWorldPoint(new Vector3(1, mashtab, 0));

Debug.Log($"B:{bottomLeft}, T:{topRight}");

Debug.DrawLine(bottomLeft, topRight);

// Создаем объекты в случайных точках в пределах видимости камеры

for (int i = 0; i < objectCount; i++)

{

Vector3 spawnPosition = Vector3.zero;

while (true)

{

// Генерируем случайные координаты в пределах видимости камеры

float spawnX = Random.Range(bottomLeft.x + 30, topRight.x - 30);

float spawnY = Random.Range(bottomLeft.y + 30, topRight.y - 30);

spawnPosition = new Vector3(spawnX, spawnY, transform.position.z);

Vector3 displacement = spawnPosition - transform.position;

if (Physics.Raycast(transform.position, displacement, out RaycastHit hit, displacement.magnitude, RockMask))

{

continue;

}

if ((spawnPosition - Submarin.position).magnitude > 50)

{

break;

}

}

// Создаем объект в найденной позиции

GameObject Boshmak = Instantiate(botinockPrefab, spawnPosition, Quaternion.identity);

if (Povorot)

Boshmak.transform.eulerAngles = new Vector3(0, 0, Random.Range(0, 360)); //поворот ботиночка

// Добавляем позицию в список созданных объектов

spawnedPositions.Add(spawnPosition);

}

// Устанавливаем флаг, что объекты созданы

hasSpawned = true;

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class MouseClickHandler : MonoBehaviour

{

public Transform submarineCenter; // Центр субмарины

public GameObject cylinderPrefab; // Префаб цилиндра

private Vector3? lastPoint = null; // Последняя точка клика (null до первого клика)

private bool isClickEnabled = false; // Флаг, указывающий на то, что клики можно обрабатывать

private List <Material> materials = new List<Material>();

private List <GameObject> cylinders = new List<GameObject>();

private LayerMask Hitmack;

//слой камешков

private LayerMask RockMask;

void Start()

{

RockMask = LayerMask.GetMask("ROCKS");

Hitmack = LayerMask.GetMask("Default");

// Сохраняем время запуска программы

StartCoroutine(EnableClicksAfterDelay(4f)); // Включаем клики через 4 секунды

}

void Update()

{

// Проверяем, можно ли обрабатывать клики

if (!isClickEnabled) return;

// Проверяем нажатие левой кнопки мыши

if (Input.GetMouseButtonDown(0))

{

// Создаём луч из экрана в мир

Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

RaycastHit hit;

// Если луч попадает в объект с коллайдером

if (Physics.Raycast(ray, out hit, 1000, LayerMask.GetMask("Water")))

{

Vector3 hitPoint = hit.point; // Точка, куда попал луч

if (lastPoint == null)

{

// Если это первый клик, строим от субмарины

CreateCylinder(submarineCenter.position, hitPoint);

}

else

{

// Если это не первый клик, строим от последней точки

CreateCylinder(lastPoint.Value, hitPoint);

}

// Сохраняем текущую точку как последнюю

}

}

}

IEnumerator EnableClicksAfterDelay(float delay)

{

// Ждём указанное количество секунд

yield return new WaitForSeconds(delay);

// Разрешаем обрабатывать клики

isClickEnabled = true;

}

void CreateCylinder(Vector3 startPoint, Vector3 endPoint)

{

// Вычисляем направление и длину

Vector3 direction = endPoint - startPoint;

float distance = direction.magnitude;

Vector3 displacement = startPoint - endPoint;

if (Physics.Raycast(endPoint, displacement, out RaycastHit hit, 5000, RockMask))

{

return;

}

// Создаём цилиндр

GameObject cylinder = Instantiate(cylinderPrefab);

materials.Add(cylinder.GetComponent<MeshRenderer>().material);

cylinders.Add(cylinder);

// Устанавливаем позицию (середина между начальной и конечной точками)

cylinder.transform.position = startPoint + direction / 2;

// Направляем цилиндр, но с учётом оси Y

cylinder.transform.up = direction.normalized;

// Настраиваем длину и ширину цилиндра

cylinder.transform.localScale = new Vector3(3, distance / 2, 0.3f);

if (Physics.SphereCast(startPoint, 2f, direction, out hit, distance, Hitmack))

{

if (hit.transform.TryGetComponent<Chest>(out Chest Chest))

{

Colorize(Color.green);

Debug.Log("ПОБЕДА!");

}

else if(hit.transform.TryGetComponent<Boots>(out Boots Boots))

{

foreach (var cyl in cylinders)

Destroy(cyl);

Colorize(Color.red);

materials.Clear();

cylinders.Clear();

lastPoint = null;

Debug.Log("НЕУДАЧА!");

}

}

else

{

lastPoint = endPoint;

}

}

void Colorize(Color color)

{

foreach (var material in materials)

material.color = color;

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Movement : MonoBehaviour

{

private Rigidbody rb;

public Transform targetPosition;

public float speed = 5f;

void Start()

{

rb = GetComponent<Rigidbody>();

}

private void FixedUpdate()

{

// Рассчитываем направление к целевой позиции

Vector3 direction = (targetPosition.position - transform.position).normalized;

// Рассчитываем новое положение на основе скорости

Vector3 newPosition = transform.position + direction \* speed \* Time.fixedDeltaTime;

// Ограничиваем перемещение, чтобы не пройти мимо цели

if (Vector3.Distance(newPosition, targetPosition.position) < speed \* Time.fixedDeltaTime)

{

newPosition = targetPosition.position;

}

// Перемещаем объект с помощью MovePosition

rb.MovePosition(newPosition);

}

}